nuBOINC: BOINC Extensions for Community Cycle Sharing

Patrick Johann Pircher
Under supervision of Prof. João Nuno de Oliveira e Silva
Dep. INEESC, IST, Lisbon, Portugal

November 27, 2014

Abstract

At the present time, it is difficult for users to benefit from cycle sharing over the Internet, even with well known infrastructures such as BOINC, since it is difficult for an ordinary user to install the required infrastructure, develop the processing applications and gather enough computer cycle donors.

In general, computer owners only have one role in the process: to donate their computers processing power. In this paper we describe a set of BOINC extensions that allow any user to create and submit jobs that can take advantage of remote idle cycles.

In order to submit their jobs, users only have to provide the input files, define the processing application as well as the command line that will be provided to the application. Later, users will contact the server, receive a set of jobs, and process them in a secure virtual machine. These users can subsequently take advantage of other people's computer cycles. This system allows a exhaustive definition of jobs while leveraging a cycle-sharing platform into a global computer cycle market.

Keywords: BOINC, volunteer computing, grid computing, P2P

1 Introduction

Berkeley Open Infrastructure for Network Computing (BOINC) [1] is a platform for the distribution of parallel jobs to be executed on remote computers. In fact some computer user communities could take advantage of remote idle cycles to speed up their jobs, but do not have the skills to efficiently use BOINC. For instance, users can be non specialists or designers who use ray tracing software to render movies or images, or researchers that use statistical softwares to process very large data sets. In order to use BOINC, one has to setup the BOINC server, develop the data processing application and create the data sets to be processed. After these preparations, donors can register themselves to provide their computer's processing power to the project. The problem emerges when normal users want to create their own project. They need to have a high profile in order to attract enough donors and they also need to know a lot about C++ programming. Projects that cannot satisfy these requirements will not be able to take advantage of available remote cycles. Benefits will be low for short-term projects, and for projects not capable of attracting donors. To allow users to create jobs in a cycle-sharing system, they should be allowed to use the applications or programming languages they know, and there should immediately be enough cycle donors to speed up their jobs.

BOINC already promotes the use of a virtual machine as job execution environment, which reduces the code development cost, as users do not have to learn a new programming language (C or Java) or adapt the code for various different Operating Systems (OS). Furthermore, the execution environment will be entirely separated from the host computer, allowing users to install any software needed and at the same time assure that the host computer will not be compromised.

In this paper we present extensions to BOINC that allow every user to submit and execute jobs. Users will have two roles: on one hand as the own-

ers of jobs that are to be executed on remote computes, and as owners of the computers where jobs will be executed. In order to accomplish this, a few changes to the BOINC server software were made: Development of a custom BOINC application, additional server software, browser client and a manager that receives the result. The project was named nuBOINC. After donating cycles in nuBOINC, a computer owner will receive credits and be able to take advantage of remote cycles to speed up his jobs. This compels users to provide more and more cycles to others and can be further promoted by giving users with a high amount of credits a higher initial priority when they need remote cycles. Complementary, users with a low amount of credits will also have low initial priority. In the event that numerous users with high priority create many tasks, a system is needed that does not cause a total preemption of low priority tasks. To guarantee fairness, the actual rank will be obtained by using the time difference since the creation of the task plus the initial priority, consequently tasks that were created some time ago can be ranked higher than newly generated tasks. This new relationship within the system will increase the number of users and the amount of time users will want to share their computer with remote users.

As shown in Figure 1 there are two roles when interacting with our extended BOINC server:

- Users that submit jobs and that receive the results though the nuBOINC Manager (user1): Project creators.

- BOINC users that execute the jobs (user1 and user2): Donors provide they computers' CPU cycles to the nuBOINC server to process jobs. Users can be both donors and project creators.

To submit and create new jobs(1), project creators can use the developed website which allows easy creation of bag-of-tasks or the REST-API which gives much more control but is for advanced users.

The server waits for BOINC client requests to distribute each job (2). When contacted, it checks which user is requesting work and selects jobs according to this information.

Client receiving the job information invoke our application which downloads required files using the BitTorrent protocol and executes the job (3). Finally, the results will be reported (4) and received by the project creator (5).

From preliminary experiments, we can conclude that these extensions allow the definition and execution of many different kinds of jobs, all of them taking advantage of remote idle cycles. We managed to execute a batch of video rendering, adding effects to the video parts, as well as running simple Python scripts. In the next section, we present existing cycle-sharing platforms and how they relate to our proposed solutions. In the following sections, we present the extensions made to BOINC and their evaluation. Finally, we present the conclusions and future enhancements.

2 Related Work

2.1 Distributed Computing

BOINC is the best known platform for the creation and execution of distributed computing projects, providing all the data storage, communication and client management infrastructure. The project manager has to program a C++ application that will be executed on the client computers to process data. Even though it is easy to install project hosting infrastructures, two issues arise: it is necessary to have C++ programming knowledge to create the applications and it is necessary to publicize the project so as to attract clients. BOINC wrappers [2] allows the use of legacy applications to process tasks in a BOINC project. Project developers only have to define the configuration file where it says how the legacy application will be executed. Both the wrapper and the legacy application are downloaded by the client, and when executed, the wrapper invokes the legacy application.

Even with this solution, short-term projects or those without capacity to obtain cycle donors cannot take advantage of BOINC. BURP [3] and Leiden Classical [4] are BOINC Projects that allow users to submit their own input files to be processed. However users are unable to submit their own processing software. A great deal of research has been done into systems that allow users to submit jobs [5], those are either not applicable to public cycle sharing systems or they are neither currently publicly available nor hosted anywhere. For instance, CONDOR [6] allows the execution of submitted jobs on remote computers on a LAN. But it assumes that all attached computers are

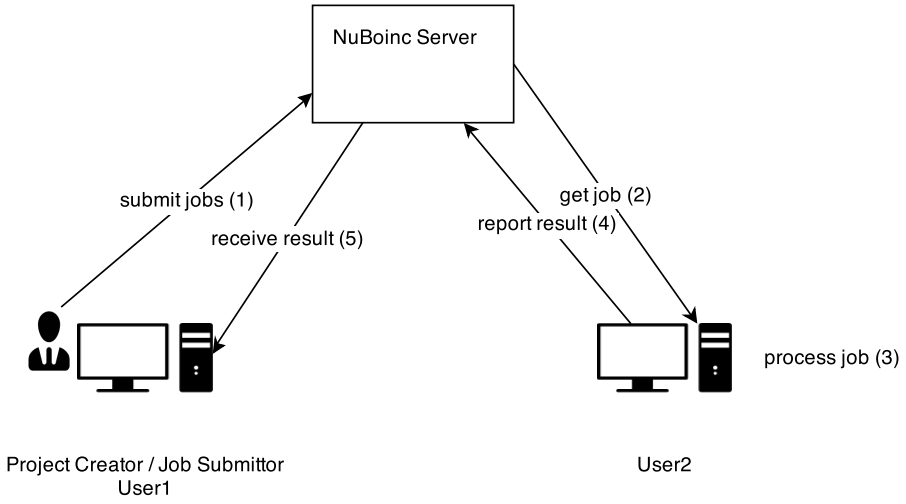

Figure 1: Extended BOINC

trusted and are constantly connected, which does not apply to volunteer computing. Another interesting system is CompuP2P which uses a decentralized system to distribute the tasks, though like other researched systems [5] no public software is available.

POPCORN [7] and G2-P2P [8] provide a programming API that developers must integrate directly into their application in order to use remote cycles. Although both did exist they are not maintained anymore This leaves BOINC as the most successful system, which comes "from the use of two natural human reactions: empathy with the problem being solved, and competitiveness among users" [5].

With this in mind, we extend BOINC so that users already supporting other projects can easily join nuBOINC. Given that the incentive for solving a specific problem does not exist in this project, another system must be used. Peer-to-Peer systems also provide incentives to users by limiting resources to free riders and giving more resources to good users. This can be applied to nuBOINC, free riders being users who only submit jobs but are reluctant to process any from the system. In BitTorrent this is solved by using a Choking Algorithm [9] which is based on the tit-for-tat policy [1].

Then again P2Pis different to nuBOINC in some aspects: In general, P2P is decentralized and the time period in which the incentive algorithms can be applied is limited (until the download completes), whereas in nuBOINC a central server is present and the time period for an incentive algorithm can be infinite. This could possibly limit the applications of P2P incentive algorithms. Problems such as free-riding and Sybil attacks [2] on P2P systems are common in such systems and some solutions have been proposed [10, 11]. The conclusion is that the current mechanisms are not sufficient and a trustworthy/more reliable entity is needed. This entity is called a Super Node and manages a currency and reputation system. It has also been mentioned that using currency has the side-effect that users would be able to accumulate currency for future use. Furthermore, it is just as important to prevent users from being able to achieve and maintain a dominant position. Even users without malicious intent contributing to the system will raise their amount of currency indefinitely and consequently dominate the system.

Therefore currency and reputation need to be limited in order to guarantee balance. This could be applied to nuBOINC. Here, the Super Node is the central server and credits represent the currency—even though this is not as dynamic as in a P2P system, since only one server is available. Similarly, the issues surrounding BitTorrent integration into nuBOINC have been researched [12, 13]. However the assumptions are that the files are located on the server and each torrent corresponds to one file. This does not apply to nuBOINC. Although this may be true, some general issues still remain. Those are the fire-

[1] tit-for-tat is an English saying meaning "equivalent retaliation"

[2] The Sybil attack is an attack where a user creates many identities to take advantage of a reputation system.

wall/router configuration needed for torrent to work and malicious users who try to send bad data through torrent.

2.2 Virtual Machines

A virtual machine is a software implementation of a physical computer. It supports the execution of a complete operating system in an isolated space.

We will use this property to run any application in a safe way as well as making it easy for developers and project creators as they only need to target one operating system. There is a great deal of different software that enables the creation of virtual machines, e.g. VirtualBox [14], VMware [15], KVM[16] and XEN[17].

In this paper, we propose the usage of VirtualBox. Not only is it already included in the BOINC installer, it also provides an easy API and some important features.

First of all, VirtualBox runs on all the well known operating systems like Windows, Mac and Linux, and it provides the functionality to save and resume the machine state. Additionally, it allows marking a medium as immutable [18], consequently saving all changes to a different location.

This feature in particular allows the creation of multiple machine instances with the same disk but with different processes running on each instance. Ultimately, these features are used to save space and time when executing BOINC tasks.

3 Architecture

3.1 Boinc Integration

The infrastructure architecture can be divided into various parts as shown in Figure 2. The newly developed components are: server application, javascript web application, nuBOINC Manager and the nuBOINC application. They were developed in order to allow the use of the original BOINC-Client so that donors do not need to install a new one. The server code has only a few lines changed in order to guarantee fairness as described in section 4. In order to allow the execution of user submitted jobs, a BOINC application and an additional server application was developed. The nuBOINC application communicates with the newly provided server through a REST-based API. The original BOINC web interface was extended with a javascript web application which allows user registration, user login, project creation and job submission. Since the web interface now allows more control over workunits, security has been increased by providing temporary API-Tokens on login.

A user that requires remote computing, can use the website to create a project and submit jobs (1). Afterwards all required files need to be hosted using the nuBOINC Manager (2).

When a donor requests a job, it will receive information regarding the job files, processing software, arguments and virtual machine (VM) configuration (3). The BOINC Client then downloads the specified VM (4) and afterwards the required job files (5). All are transferred using the BitTorrent protocol. The virtual machine will then be started, all requirement will be installed and subsequently the job will be executed (6) . Later, the files will be send directly to the project creator (7). Finally, the job finishes and the BOINC Client reports a successful execution (8).

Some modifications were made with respect to the job information organization within the server. In a normal BOINC setup, work is grouped into projects and all jobs from the same project are executed by the same application. With this extension, all user submitted jobs are processed within the same BOINC project and by the same application, but they belong to different user projects. For each job there is one workunit (input files and execution parameters) and several replicas of each workunit, called results. With this extension, the workunit input file is the job description and torrent file for the nuBOINC application, which then selects and downloads the input files and executes the submitted job.

The output files are not sent back to the server. Instead, they are sent directly to the project creator. After executing a job, our application will try to obtain an address from the server, to which it should send the output files. If no address is available at that time, or if the transmission fails, our application will retry every 10 minutes. In order to prevent someone from obtaining the address and sending malicious files, an authentication system was put in place. The nuBOINC application has to send the values of *authenticator*, *hostid*, *userid*, *result_name* and *wu_name*. These values are provided by APP_INIT_DATA [19] and if the combination is not correct, the server will

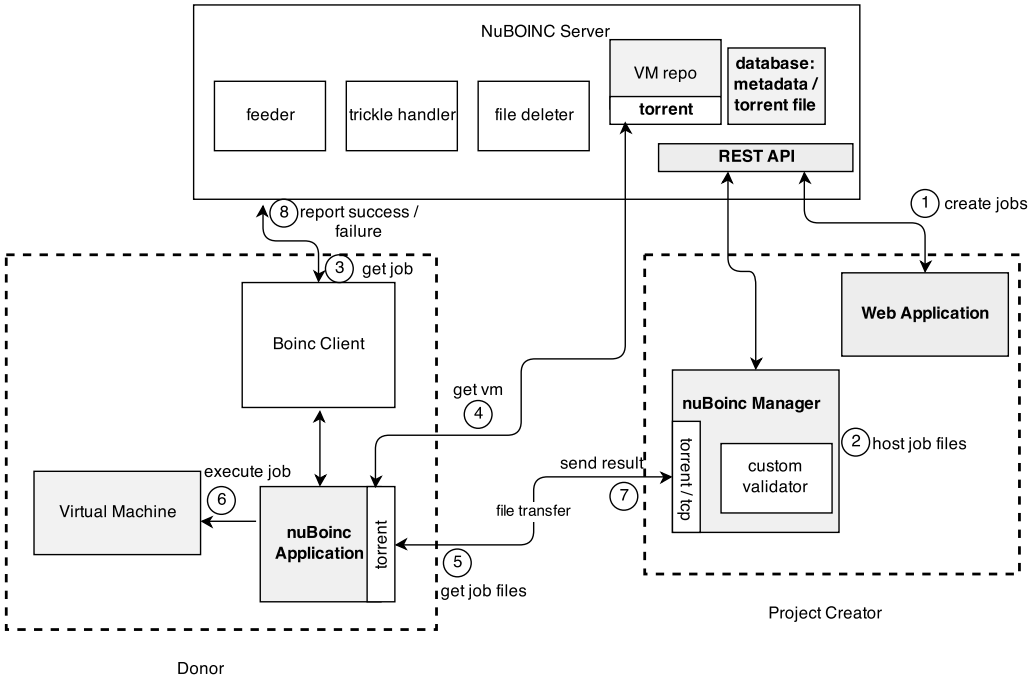

Figure 2: Extended BOINC architecture

not respond.

3.2 nuBOINC Manager

The main purpose of the nuBOINC Manager is to receive the output files generated by the various remote clients. When the Manager is started, it will ask for the user's authentication and login into our server. It will retrieve the information about the user's projects and jobs and display them. At the same time, it creates a local server on a specific port, informs the server about it and tries to forward the port on the local router trough UPnP [20] if the service is available. The server receives the port and retrieves the IP number. This information is kept on the server for three hours. The manager resends the port every 60 minutes in case the connection was lost and the IP number changed. This data is sent to the nuBOINC application when it needs to send the results.

The nuBOINC Manager also features torrent creation with a special attribute: It adds pad files between the torrents. This allows any torrent client to individually select files without downloading parts of other files. The reason is explained in [21]: "clients do not transfer files by files but by pieces which have the same size. At the creation time of a torrent, all the files in the torrent are virtually concatenated and then the resulting data chunk is split into evenly sized pieces, of a size appointed by the torrent making application and indicated in the torrent file. If there is more than one file in the torrent, there might be some pieces containing the end of file A and at the same time the beginning of file B." Therefore, if A is selected B will also be selected and allocated on the file system and therefore use up more space.

3.3 Server and Client

The new server and JavaScript application were implemented with an eye on overall performance improvements and server load reduction. The server provides a REST API, which is consumed by the Javascript Web Application, the nuBOINC Manager and the developed nuBOINC application. As more users begin to use nuBOINC, server load increases. A server using PHP with Apache will create a thread for each request, PHP is then invoked in those threads. This causes a big overhead if many connections are being made and the time to process a request is short. Our REST API only transfers small amounts of data at a time and has low processing times. Therefore the server is implemented in Python using the Gevent [22] library. Gevent provides pseudo-threads, the program is processed sequentially, but the context switches when writ-

ing or reading file like objects. This provides high-concurrency and therefore decreases latency if many connections are made to the server, since accepting new connections creates no big overhead. This and the fact that no files are saved remotely allows the use of small and inexpensive servers.

4 Scheduler and Feeder

When contacted by a BOINC-Client, the adapted server prioritizes jobs from users who are requesting workunits. If no user jobs are found, the workunit with the highest priority is returned. The priority is calculated using the time difference from when it was created and the user's total credit. The function used is based on the inverse exponential decay formula as shown in eq 2. The result is always between 0 and 100. The factor 5 was selected to get 100 when a user's credit equals the maximum credit. Each day increments the priority by 10 so that after at least 10 days the priority of the workunit is at 100. This is the default setting, but it can be modified. The priority may increase further if the job is not processed at that time. To prevent accumulation of credits and obtaining an dominant position, the maximum credit constant is determined by the nuBOINC hoster. After a certain time, the credits are reset to zero. The maximum credit value should reflect the average credits obtained in a certain time period. Assuming that the current average credit per day is 2000 and a time period of about three months, the maximum credit could be set to 200000. This function translates into the following:

- all users with a total credit close to the maximum will begin with a high priority but less competition between each other as changes to the total credit will not have a great effect on the priority .
- all new users with low credits will have high competition between each other and receive great a increase in priority for a raise in their credits.
- a new submitted job from a user with a high priority may have a lower priority than a job from a low priority user that was created some time ago.
- the Sybil attack is useless, since users always start with zero credits.
- free-riders do not receive credits and are penalized with possible long wait times.
- prevents accumulation of credits over longer time periods and dominant positioning.

$$\tau = maximumcredit \tag{1}$$

$$priority = round(100 * (1 - e^{-user_total_credit/\tau * 5})) \tag{2}$$

Assume the example given in Table 1. Three PCs exists in the network, each belonging to a respective user. In *example 1* all submit 100 jobs but only PC A is processing tasks. Since A is processing tasks and there are jobs submitted by A those will be selected, ignoring priorities. After all tasks from A are processed, A will begin to execute tasks from others. As shown in *example 2*, the priorities will now be considered by the scheduler. B has a higher priority than C and therefore his jobs will be selected.

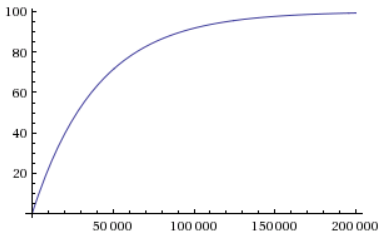

Figure 3: Inverse Exponential Decay Formula; with maxcredit=200.000

Figure 4 shows the behavior of the scheduler over a time period. On day 1 the task *B1* was selected and executed. Later B creates a new task which receives the priority 30. On the second day task *C1* is selected by having a higher priority than *B2*. Later C creates a new task which receives the priority 70. In this instant, the task *B2* was created 5 days ago. It had an initial priority of 30 which increases each day by a total amount of 10. Consequently the actual priority is now 80, which is higher than the rank of *C2* and thus is selected for processing.

	Example 1			Example 2		
PCs	A	B	C	A	B	C
#jobs	100	100	100	0	100	100
Priority	15	50	30	/	50	30
Workers		A			A	
Prioritize		A			B	

Table 1: Example of task selection

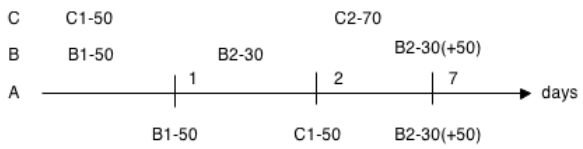

Figure 4: Example of task selection over a time period

4.1 Job execution

Since users can submit any type of program, we need to prevent malicious actions. Those include influencing the computer where the script is being executed and any action through the network. Therefore all jobs are run inside a virtual machine. This separates the job's environment from the actual computer. After installing all necessary applications, the network to and from the virtual machine will be isolated. This also allows to run the job on any Operating system that supports virtual machines. Furthermore, snapshots can be taken to pause the job and resume at a later time. Since the Operating System inside the virtual machine is an Ubuntu derivate the *apt-get* command is used to install necessary packages. First the command *apt-get update* is executed to refresh the package list, afterwards *apt-get install* x is executed. A shared folder between the Host and the Guest will be generated, that points to the downloaded torrent folder. This will be used as execution environment. The project creator is responsible for choosing the correct combination of the Ubuntu version and packages to be installed.

4.2 File transmission

The VirtualBox image and the input files are downloaded using the BitTorrent protocol [23]. The image is kept in a separate location and can be used by all jobs at the same time by marking it immutable. This instructs VirtualBox to keep the changes in a different location and allows us to download the file only once. The image is provided by our server whereas the input files must be supplied by the project creator. He needs to create and seed the torrent, which can be done with the nuBOINC Manager. All input files are listed in the torrent file as defined by the BitTorrent protocol [23] and our application downloads the required ones. During the jobs execution, the torrent will continue to be seeded to take full advantage of the BitTorrent protocol. By using the variant type "Stream" for the VirtualBox disk image the size was reduced to around 400 Megabytes from 1.4 Gigabytes, hence reducing transmission and disk space.

4.3 Transmission Overhead

The only part that can significantly increase the time to run a task is the download of the disk image and input files. The disk image only needs to be downloaded once, therefore the time lost in the download only applies to the first tasks: the one which is downloading it and the ones which are waiting for the download to finish. Subsequent tasks do not need to download it again. The process needs to be repeated only when the temporary folder is cleaned. This applies to all tasks, even from different users. Likewise, an input file may be downloaded once but can be used by multiple tasks. This happens when one task downloads the input file and then another task also needs the same file. Since the files are being seeded by the first task on the same computer, the second task can obtain it much faster.

4.4 Usability

The intention of nuBOINC was to create an easy to use interface with which project creators can submit tasks. Here we list what project creators need to know beforehand and the steps taken to submit a job. Basically users need to know how to invoke their application. They should be able to test the needed application on the OS and determine which packages and repositories are required. They do not need to now how to install the packages, this is done automatically by nuBOINC. When submitting a task, the following steps have to be taken:

1. create an account on the website
2. download the nuBOINC Manager

3. generate a torrent file using the nuBOINC Manager, including all input files and custom executables

4. use the website to upload the torrent and specify the software to be used and/or the executable provided

5. define the arguments (if applicable) for the application and the names of the output files

6. select the Operating System (different versions of Ubuntu) to be used and the packages/repositories required

7. set additional job properties like disk usage and memory usage

After submitting the tasks, users should leave the nuBOINC Manager running and connected to the Internet in order to receive results. Assuming users know Ubuntu and how to invoke an application through the command line, the process is straightforward, the nuBOINC Manager being the only new software they must interact with and having only two functions: creating torrents and receiving results. In conclusion, the users can, after a small introduction, work quickly and independently.

5 Evaluation

In order to evaluate the usability, we deployed our server and tested the definition and execution of jobs. We did 3 experiments with different input files, outputs and arguments. For each test, we created a torrent containing the input file names and possible executables or scripts. The first experiment performed consisted of a simple Python script reading one file, converting all characters to uppercase and writing them to an output file. After we had uploaded the torrent to our server we marked the script as executable, selected python as the executable's caller and let the website create a job for each input file. In order to test multiple input and output files, the previous application was adapted so that it applies the same process to two input files and writes the result to two output files. Furthermore, it was run with the Python interpreter PyPy, which demonstrates that the defined package PyPy was installed. In the last test, various video parts needed to be processed with avconv. On our website, we defined two packages that are required by avconv and then let it create a task for each file.

Additionally, the correct credit granting, job scheduling and prioritization was verified. In conclusion, all experiments yield the expected results:

- All tasks were executed successfully and all results were transmitted back to the nuBOINC Manager.
- Users will first process they own jobs, but not receive any credit
- Users with many credits receive a higher prioritization for their jobs and users with scarce credits receive a low prioritization for they jobs
- The priority of jobs increase over time

6 Conclusion and Future Work

We managed to extend the original BOINC infrastructure into a system that allows the execution of user submitted jobs. These jobs can be configured individually using the REST API. A job submission user interface was added to BOINC, allowing the definition of each job's input files and parameters without any programming knowledge. By allowing the execution of any application that can run on Linux, users do not have to develop the BOINC applications to be executed on remote computers and they do not need to deal with BOINC integration. These applications run in a secure virtual machine, protecting the host computer from malicious code. This should be extended to save the machine state when the system is shut down and can be further improved by making snapshots after software was installed so it can be reused. The REST API gives almost direct access to the BOINC Work Creator. This makes the jobs highly configurable and allows anyone with programming knowledge to setup a specialized work generator. To guarantee some fairness, scheduling on the server was modified. In the event that the owner of a client has jobs to be executed, the scheduler sends his jobs first. Moreover, a prioritizing system was implemented that prefers jobs from users with a high amount of credits or jobs that were created earlier. Additionally, the credit system should be improved and it is also equally important to further investigate the security aspects of this infrastructure.

References

[1] D. P. Anderson and G. Fedak, "The computational and storage potential of volunteer computing," in *Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06.*, vol. 1. IEEE, 2006, pp. 73–80.

[2] D. P. Anderson. (2014) The boinc wrapper. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/WrapperApp

[3] (2014) big ugly rendering project. [Online]. Available: http://burp.renderfarming.net/

[4] U. of Leiden. (2014) Leiden classical. [Online]. Available: http://boinc.gorlaeus.net/

[5] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang, "Characterizing and classifying desktop grid," in *Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007.* IEEE, 2007, pp. 743–748.

[6] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," in *Distributed Computing Systems, 1988., 8th International Conference on.* IEEE, 1988, pp. 104–111.

[7] N. Nisan, S. London, O. Regev, and N. Camiel, "Globally distributed computation over the internet-the popcorn project," in *18th International Conference on Distributed Computing Systems, 1998. Proceedings.* IEEE, 1998, pp. 592–601.

[8] R. Mason and W. Kelly, "G2-p2p: a fully decentralised fault-tolerant cycle-stealing framework," in *Proceedings of the 2005 Australasian workshop on Grid computing and e-research-Volume 44.* Australian Computer Society, Inc., 2005, pp. 33–39.

[9] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[10] P. D. Rodrigues, C. Ribeiro, and L. Veiga, "Incentive mechanisms in peer-to-peer networks," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW).* IEEE, 2010, pp. 1–8.

[11] T.-Y. Wu, W.-T. Lee, N. Guizani, and T.-M. Wang, "Incentive mechanism for p2p file sharing based on social network and game theory," vol. 41. Elsevier, 2014, pp. 47–55.

[12] F. Costa, L. Silva, I. Kelley, and I. Taylor, "Peer-to-peer techniques for data distribution in desktop grid computing platforms," Tech. Rep., 2008.

[13] F. Costa, L. Silva, G. Fedak, and I. Kelley, "Optimizing the data distribution layer of boinc with bittorrent," pp. 1–8, 2008.

[14] (2014) Virtualbox. [Online]. Available: https://www.virtualbox.org/

[15] (2014) Vmware. [Online]. Available: http://www.vmware.com/

[16] (2014) Kernel based virtual machine. [Online]. Available: http://www.linux-kvm.org/

[17] (2014) Kernel based virtual machine. [Online]. Available: http://www.xenproject.org/

[18] O. Corporation, "Virtualbox user manual special image write modes," 2014.

[19] D. P. Anderson. (2013) Startup and status data. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/StatusApi

[20] U. Forum. (2008) Universal plug and play (networking protocols). [Online]. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf

[21] BitComet. (2010, November) Align file to piece boundary. [Online]. Available: http://wiki.bitcomet.com/align_file_to_piece_boundary

[22] D. Bilenko *et al.* (2014) Gevent. [Online]. Available: http://www.gevent.org/

[23] B. Cohen. (2012) The bittorrent protocol specification. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html